

**The Storage Resource Manager Interface Specification**  
**Version 2.1**  
**Final Candidate 2**  
**(SRM.spec.v2.1.Final.Candidate2.Oct20.doc)**  
**(Modifies SRM.spec.v2.1.Final.Candidate1.July17.doc)**

**Contributors to this document are:**

EDG-WP2: Peter Kunszt, Heinz Stockinger, Kurt Stockinger, Erwin Laure  
EDG-WP5: Jean-Philippe Baud, Stefano Occhetti, Jens Jensen, Emil Knezo,  
Owen Synge, Olof Barring  
JLAB: Bryan Hess, Andy Kowalski, Chip Watson  
FermiLab: Don Petravick, Timur Perelmutov, Rich Wellner  
LBNL: Junmin Gu , Arie Shoshani, Alex Sim

**This version prepared by:**  
Junmin Gu, Alex Sim, Arie Shoshani  
Lawrence Berkeley National Laboratory

It reflects in part decisions discussed in  
<http://sdm.lbl.gov/srm-wg/doc/SRM.v2.1.joint.func.design.doc>

**Introduction**

This document contains the interface specification of SRM 2.1. It incorporates the functionality of SRM 2.0 (see <http://sdm.lbl.gov/srm-wg/doc/srm.methods.v2.0.doc>), but is much expanded to include additional functionality, especially in the area of dynamic storage space reservation and directory functionality in client-acquired storage spaces.

This document reflects the discussions and conclusions of a 2-day meeting in December 2002, as well as subsequent email correspondence and conference calls. The purpose of this activity is to further define the functionality and standardize the interface of Storage Resource Managers (SRMs) – a Grid middleware component. This document is a follow up to the basic SRM design consideration document that describes the basic functionality of SRM Version 2.0 (see <http://sdm.lbl.gov/srm-wg/doc/SRM.v2.0.joint.func.design.doc>).

The document is organized in four sections. The first, called “Defined Structures” contain all the type definitions used to define the functions (or methods). The next 3 sections contain the specification of “Space Management Functions”, “Directory Functions”, and “Data Transfer Functions”. All the “Space Management Functions”, “Directory Functions” are newly added functions, and “Data Transfer Functions” are slightly modified versions of the SRM V2.0 specification.

It is advisable to read the document SRM.v2.1.joint.func.design.doc posted at <http://sdm.lbl.gov/srm-wg> before reading this specification, since the reasoning for the decisions reflected in this specification are described there in detail.

### Meaning of terms:

By “https” we mean http: protocol with GSI authentication. At this time, any implementation of http with GSI authentication could be used. It is advisable that the implementation is compatible with Globus Toolkit 3.0 or later versions.

- Primitive types used below are consistent with XML build-in schema types: i.e.
  - long is 64bit: (+/-)9223372036854775807
  - int is 32 bit: (+/-)2147483647
  - short is 16 bit: (+/-)32767
  - unsignedLong ranges (inclusive): 0 to 18446744073709551615
  - unsignedInt ranges (inclusive): 0 to 4294967295
  - unsignedShort ranges (inclusive): 0 to 65535
- The definition of the type “anyURI” used below is compliant with the XML standard. See <http://www.w3.org/TR/xmlschema-2/#anyURI>. It is defined as: "The lexical space of anyURI is finite-length character sequences which, when the algorithm defined in Section 5.4 of [XML Linking Language] is applied to them, result in strings which are legal URIs according to [RFC 2396], as amended by [RFC 2732]".
- In “localSURLInfo”, we mean local to the SRM that is processing the request.
- TStorageSystemInfo is added in the arguments of functions srmPrepareToGet() srmPrepareToPut() and srmCopy(). This is to simplify the case when all files sent to the request share the same storageSystemInfo. If storageSystemInfo is provided at the request level and the file level, SRM will use the one provided at the file level.

## Namespace SRM:

**Notation:** underlined attributes are *REQUIRED*.

<i>Defined Structures:</i>
----------------------------

enum	<b>TSpaceType</b>	{ Volatile, Durable, Permanent }
enum	<b>TFileStorageType</b>	{ Volatile, Durable, Permanent }
enum	<b>TFileType</b>	{ File, Directory, Link }
enum	<b>TPermissionMode</b>	{ NONE, X, W, WX, R, RX, RW, RWX }
enum	<b>TPermissionType</b>	{ ADD, REMOVE, CHANGE }
enum	<b>TRequestType</b>	{ PrepareToGet, PrepareToPut, Copy }
enum	<b>TOverwriteMode</b>	{ Never, Always, WhenFilesAreDifferent }
typedef	string	<b>TRequestToken</b>
typedef	string	<b>TSpaceToken</b>
typedef	string	<b>TUserID</b>
typedef	string	<b>TGroupID</b>
typedef	TPermissionMode	<b>TOwnerPermission</b>
typedef	struct { TUserID TPermissionMode	UserID, mode } <b>TUserPermission</b>
typedef	struct { TGroupID TPermissionMode	GroupID, mode } <b>TGroupPermission</b>
typedef	TPermissionMode	<b>TOtherPermission</b>
typedef	string	<b>TChecksumType</b>
typedef	string	<b>TChecksumValue</b>
typedef	unsigned long	<b>TSizeInBytes</b>
typedef	dateTime	<b>TGMTTime</b> // format is same as in XML dateTime type, except // no local time extension is allowed. // e.g. 1999-05-31T13:20:00 is ok, // (for 1999 May 31st, 13:20PM, UTC) // but 1999-05-31T13:20:00-5:00 is not.
typedef	unsigned long	<b>TLifeTimeInSeconds</b>
typedef	anyURI	<b>TSURL</b> // site URL

```

typedef      anyURI      TTURL // transfer URL

typedef      struct {string      path, // both dir and file
      TReturnStatus      status,
      TSizeInBytes      size, // 0 if dir
      TOwnerPermission      ownerPermission,
      TUserPermission[]      userPermission,
      TGroupPermission[]      groupPermission,
      TOtherPermission      otherPermission
      TGMTTime      createdAtTime,
      TGMTTime      lastModificationTime,
      TUserID      owner,
      TFileStorageType      fileStorageType,
      TFileType      type, // Directory or File
      TLifeTimeInSeconds      lifetimeAssigned,
      TLifeTimeInSeconds      lifetimeLeft,
      TCheckSumType      checkSumType,
      TCheckSumValue      checkSumValue,
      TSURL      originalSURL, // if path is a file
      TMetaDataPathDetail[]      subPath // optional recursive
    } TMetaDataPathDetail

typedef      struct {TSpaceType      type,
      TSpaceToken      spaceToken,
      Boolean      isValid,
      TuserID      owner,
      TSizeInBytes      totalSize, // best effort
      TSizeInBytes      GuaranteedSize,
      TSizeInBytes      unusedSize,
      TLifeTimeInSeconds      lifetimeAssigned,
      TLifeTimeInSeconds      lifetimeLeft
    } TMetaDataSpace

typedef      string      TStorageSystemInfo
// TstorageSystemInfo can contain but is not limited to the following:
// storage device, storage login ID, storage login authorization.

typedef      struct {Boolean      isSourceADirectory,
      Boolean      allLevelRecursive, // default = false
      int      numOfLevels // default = 1 } TDirOption

typedef      struct {TSURL      SURLOrStFN,
      TStorageSystemInfo      storageSystemInfo } TSURLInfo

typedef      struct {TSURLInfo      fromSURLInfo,
      TLifeTimeInSeconds      lifetime, // pin time

```

```

        TFileStorageType      fileStorageType,
        TSpaceToken           spaceToken,
        TDirOption            dirOption
    } TGetFileRequest

typedef      struct {TSURLInfo      toSURLInfo, // local to SRM
                  TLifeTimeInSeconds lifetime, // pin time
                  TFileStorageType  fileStorageType,
                  TSpaceToken       spaceToken,
                  TSizeInBytes      knownSizeOfThisFile
    } TPutFileRequest

typedef      struct {TSURLInfo      fromSURLInfo,
                  TSURLInfo        toSURLInfo,
                  TLifeTimeInSeconds lifetime, // pin time
                  TFileStorageType  fileStorageType,
                  TSpaceToken       spaceToken,
                  TOverwriteMode    overwriteMode,
                  TDirOption        dirOption
    } TCopyFileRequest

```

// In TGetFileRequest, TPutFileRequest, TCopyFileRequest:

// the default value of “lifetime”:

//       for Volatile or Durable files will be the lifetime left in the space

//       of the corresponding file type.

// the default value of “fileType” is Volatile.

// The following SRM status codes are explained at the end of this document.

```

enum      TStatusCode      { SRM_SUCCESS,
                               SRM_FAILURE,
                               SRM_AUTHENTICATION_FAILURE,
                               SRM_UNAUTHORIZED_ACCESS,
                               SRM_INVALID_REQUEST,
                               SRM_INVALID_PATH,
                               SRM_FILE_LIFETIME_EXPIRED,
                               SRM_SPACE_LIFETIME_EXPIRED,
                               SRM_EXCEED_ALLOCATION,
                               SRM_NO_USER_SPACE,
                               SRM_NO_FREE_SPACE,
                               SRM_DUPLICATION_ERROR,
                               SRM_NON_EMPTY_DIRECTORY,
                               SRM_TOO_MANY_RESULTS,
                               SRM_INTERNAL_ERROR,
                               SRM_FATAL_INTERNAL_ERROR,
                               SRM_NOT_SUPPORTED,

```

```

SRM_REQUEST_QUEUED,
SRM_REQUEST_INPROGRESS,
SRM_REQUEST_SUSPENDED,
SRM_ABORTED,
SRM__RELEASED,
SRM_FILE_PINNED,
SRM_FILE_IN_CACHE,
SRM_SPACE_AVAILABLE,
SRM_LOWER_SPACE_GRANTED,
SRM_DONE,
SRM_CUSTOM_STATUS
}

```

```

typedef struct { TStatusCode statusCode,
                string      explanation } TReturnStatus

```

```

typedef struct { TSURL          surl,
                TReturnStatus status } TSURLReturnStatus

```

```

typedef struct { TSURL          fromSURLInfo,
                TSizeInBytes   fileSize,
                TReturnStatus  status,
                TLifeTimeInSeconds estimatedWaitTimeOnQueue,
                TLifeTimeInSeconds estimatedProcessingTime,
                TTURL          transferURL,
                TLifeTimeInSeconds remainingPinTime
            } TGetRequestFileStatus

```

```

typedef struct { TSizeInBytes   fileSize,
                TReturnStatus  status,
                TLifeTimeInSeconds estimatedWaitTimeOnQueue,
                TLifeTimeInSeconds estimatedProcessingTime,
                TTURL          transferURL,
                TSURL          siteURL, // for future reference
                TLifeTimeInSeconds remainingPinTime
            } TPutRequestFileStatus

```

```

typedef struct { TSURL          fromSURL,
                TSURL          toSURL,
                TSizeInBytes   fileSize,
                TReturnStatus  status,
                TLifeTimeInSeconds estimatedWaitTimeOnQueue,
                TLifeTimeInSeconds estimatedProcessingTime,
                TLifeTimeInSeconds remainingPinTime
            } TCopyRequestFileStatus

```

```

typedef      struct {TRequestToken      requestToken,
                  TRequestType          requestType,
                  int                    totalFilesInThisRequest,
                  int                    numOfQueuedRequests,
                  int                    numOfFinishedRequests,
                  int                    numOfProgressingRequests,
                  Boolean                 isSuspended
        } TRequestSummary

typedef      struct {TSURL              surl,
                  TReturnStatus          status,
                  TPermissionType        userPermission
        } TSURLPermissionReturn

typedef      struct {TRequestToken      requestToken,
                  TGMTTime               createdAtTime
        } TRequestTokenReturn

```

**notes:**

- *StorageSystemInfo* is a string that contains the login and password required by the storage system. For example, it might have the form of login:pwd@hostname, where “:” is a reserved separator between login and pwd. If hostname is not provided, it is defaulted to what’s in the accompanying site URL or the host of SRM.
- *TMetaDataSpace* can refer to a single space of each type (i.e. volatile, durable, permanent). It does not include the extra space needed to hold the directory structures.
- Regarding file sharing by the SRM, it is a local implementation decision. An SRM can choose to share files by providing multiple users access to the same physical file, or by copying a file into another user’s space. Either way, if an SRM chooses to share a file (that is, avoid reading a file over again from the source site) the SRM should check with the source site whether the user has a read/write permission. Only if permission is granted, the file can be shared.
- The type definition *SURL* above is used for both site URL and the “Storage File Name” (*stFN*). This was done in order to simplify the notation. Recall that *stFN* is the file path/name of the intended storage location when a file is put (or copied) into an SRM controlled space. Thus, a *stFN* can be thought of a special case of an *SURL*, where the protocol is assumed to be “srm” and the machine:port is assumed to be local to the SRM. For example, when the request *srmCopy* is made, the source file is specified by a site URL, and the target location can be optionally specified as a *stFN*. By considering the *stFN* a special case of an *SURL*, an *srmCopy* takes *SURLs* as both the source and target parameters.
- The *requestToken* assigned by SRM is unique and immutable (non-reusable). For example, if the date:time is part of the *requestToken* it will be immutable.

<b><i>Function specification:</i></b>
---------------------------------------

### **Space Management Functions:**

---

**summary:**

**srmReserveSpace**  
**srmReleaseSpace**  
**srmUpdateSpace**(includes size and time)  
**srmCompactSpace:**

**srmGetSpaceMetaData:**

**srmChangeFileStorageType:**

**srmGetSpaceToken:**

**details:****srmReserveSpace:**

In:	TUserID	userID,
	TSpaceType	<u>typeOfSpace</u> ,
	String	userSpaceTokenDescription,
	TSizeInBytes	sizeOfTotalSpaceDesired,
	TSizeInBytes	sizeOfGuaranteedSpaceDesired,
	TLifeTimeInSeconds	lifetimeOfSpaceToReserve,
	TStorageSystemInfo	storageSystemInfo
Out:	TSpaceType	typeOfReservedSpace,
	TSizeInBytes	sizeOfTotalReservedSpace, // best effort
	TSizeInBytes	sizeOfGuaranteedReservedSpace,
	TLifeTimeInSeconds	lifetimeOfReservedSpace,
	TSpaceToken,	referenceHandleOfReservedSpace,
	TReturnStatus	<u>returnStatus</u>

**notes:**

- *lifetimeOfSpaceToReserve is not needed if requesting permanent space.*
- *SRM can provide default size and lifetime if not supplied.*
- *storageSystemInfo is optional in case storage system requires additional security check.*
- *If sizeOfTotalSpaceDesired is not specified, the SRM will return its default quota.*

**srmReleaseSpace:**



In:	TUserID	userID,
	TSpaceToken	<u>spaceToken</u> ,
	TStorageSystemInfo	storageSystemInfo,
	Boolean	forceFileRelease
Out:	TReturnStatus	<u>returnStatus</u>

**notes:**

- *forceFileRelease=false is default. This means that the space will not be released if it has files that are still pinned in the space. To release the space regardless of the files it contains and their status forceFileRelease=true must be specified.*
- *To be safe, a request to release a reserved space that has an on-going file transfer will return false, even forceFileRelease= true.*
- *When space is releasable and forceFileRelease=true, all the files in the space are released, even in durable or permanent space.*
- *When space is released, the files in that space are treated according to their types: If permanent, keep it. If durable, perform action at the end of lifetime. If Volatile, release it at the end of lifetime.*

**srmUpdateSpace(includes size and time)**

In:	TUserID	userID,
	TSpaceToken	<u>spaceToken</u> ,
	TSizeInBytes	newSizeOfTotalSpaceDesired,
	TSizeInBytes	newSizeOfGuaranteedSpaceDesired,
	TLifeTimeInSeconds	newLifeTimeFromCallingTime,
	TStorageSystemInfo	storageSystemInfo
Out:		
	TSizeInBytes	sizeOfTotalSpace, // best effort
	TSizeInBytes	sizeOfGuaranteedSpace,
	TLifeTimeInSeconds	lifetimeGranted,
	TReturnStatus	<u>returnStatus</u>

**notes:**

- *If neither size nor lifetime are supplied in the input, then return will be null.*
- *newSize is the new actual size of the space, so has to be positive.*
- *newLifetimeFromCallingTime is the new lifetime requested regardless of the previous lifetime, and has to be positive. It might even be shorter than the remaining lifetime at the time of the call.*

**srmCompactSpace:**

In:	TUserID	userID,
	TSpaceToken	<u>spaceToken</u> ,
	TStorageSystemInfo	storageSystemInfo,
	Boolean	doDynamicCompactFromNowOn

Out:	TSizeInBytes	newSizeOfThisSpace,
	TReturnStatus	<u>returnStatus</u>

**notes:**

- *This function is called to reclaim the space for all released files and update space size in Durable and Permanent spaces. Files not released are not going to be removed (even if lifetime expired.)*
- *doDynamicCompactFromNowOn=false by default, which implies that only a one time compactSpace will take place.*
- *If doDynamicCompactFromNowOn=true, then the space of released files will be automatically compacted until the value of doDynamicCompactFromNowOn is set to false.*
- *When space is compacted, the files in that space do not have to be removed by the SRM. For example, the SRM can choose to move them to volatile space. The client will only perceive that the compacted space is now available to them.*
- *To physically force a removal of a file, the client should use srmRm.*

**srmGetSpaceMetaData:**

In:	TUserID	userID,
	TSpaceToken[]	<u>arrayOfSpaceToken</u>
Out:	TMetaDataSpace[]	arrayOfSpaceDetails
	TReturnStatus	<u>returnStatus</u>

**srmChangeFileStorageType: (applies to both dir and file)**

In:	TUserID	userID,
	TSURLInfo[]	arrayOfPath,
	TFileStorageType	<u>desiredStorage Type</u>
Out:	TSURLReturnStatus []	<u>returnStatus</u>

**notes:**

- *Either path must be supplied.*
- *If a path is pointing to a directory, then the effect is recursive for all the files in this directory.*
- *Space allocation and de-allocation maybe involved.*

**srmGetSpaceToken:**

In:	string	<u>userSpaceTokenDescription,</u>
	TUserID	userID
Out:	TSpaceToken[]	arrayOfPossibleSpaceTokens
	TReturnStatus	<u>returnStatus</u>

**notes:**

- *If userSpaceTokenDescription is null, returns all space tokens this user owns*
- *If the user assigned the same name to multiple space reservations, he may get back multiple space tokens.*

**Permission Functions:****summary:**

**srmSetPermission:** (applies to both *dir* and *file*)

**srmReassignToUser:**

**srmCheckPermission:**

**details:****srmSetPermission: (applies to both *dir* and *file*)**

In:	TUserID	<u>userID</u> ,
	TSURLInfo	<u>path</u> ,
	TPermissionType	<u>permissionType</u> ,
	TOwnerPermission	ownerPermission <sub>1</sub>
	TUserPermission[]	userPermission <sub>1</sub>
	TGroupPermission[]	groupPermission <sub>1</sub>
	TOtherPermission	otherPermission

Out:	TReturnStatus	<u>returnStatus</u>
------	---------------	---------------------

**Notes:**

- Support for srmSetPermission is optional.
- In this version, TPermissionMode is identical to Unix permission modes.
- User permissions are provided in order to support dynamic user-level permission assignment similar to Access Control Lists (ACLs).
- Permissions can be assigned to set of users and sets of groups, but only a single owner.
- In this version, SRMs do not provide any group operations (setup, modify, remove, etc.)
- Groups are assumed to be setup before srmSetPermission is used.
- If TPermissionType is ADD or CHANGE, and TPermissionMode is null, then it is assumed that TPermissionMode is READ only.
- If TPermissionType is REMOVE, then the TPermissionMode is ignored.

**srmReassignToUser:**

In:	TUserID	<u>userID</u> ,
	string	<u>assignedUser</u> ,

TLifeTimeInSeconds	<u>lifeTimeOfThisAssignment</u> ,
TSURLInfo	path // file or dir

Out: TReturnStatus returnStatus

**notes:**

- *After lifeTimeOfThisAssignment time period, or when assignedUser obtained a copy of files through srmCopy(), the files involved are released and space is compacted automatically, which ever is first.*
- *This function implies actual lifetime of file/space involved is extended up to the lifeTimeOfThisAssignment.*
- *The caller must be the owner of the files to be reassigned.*
- *permission is omitted because it has to be READ permission.*
- *lifeTimeOfThisAssignment is relative to the calling time. So it must be positive.*
- *If the path here is a directory, then all the files under it are included recursively.*
- *If there are any files involved that are released before this function call, then these files will not be involved in reassignment, even if they are still in the space.*
- *If a compact() is called before this function is complete, then this function has priority over compact(). Compact will be done automatically as soon as files are copies to the assignedUser. Whether to dynamically compact or not is an implementation choice.*

**srmCheckPermission:**

In:	TSURLInfo[]	<u>arrayOfSiteURL</u>
	TUserID	userID,
	Boolean	checkInLocalCacheOnly // default: false

Out:	<b>TSURLPermissionReturn []</b>	arrayOfPermissions
	TReturnStatus	<u>returnStatus</u>

**notes:**

- *When checkInLocalCacheOnly=true, then SRM will only check files in its local cache. Otherwise, if a file is not in its local cache, then SRM will go to the siteURL to check the user permission.*
- *If checkInLocalCacheOnly = false, SRM can choose to always check the siteURL for user permission of each file. It is also ok if SRM choose to check its local cache first, if a file exists and the user has permission, return that permission. Otherwise, check the siteURL and return permission.*

## Directory Functions:

---

**summary:**

srmMkdir:  
srmRmdir: (applies to *dir*)  
srmRm: (applies to *file*)

**srmLs: (applies to both *dir* and *file*)**  
**srmMv: (applies to both *dir* and *file*)**

**details:**

**srmMkdir:**

In:	TUserID	userID,
	TSURLInfo	<u>directoryPath</u>
Out:	TReturnStatus	<u>returnStatus</u>

**notes:**

- *Consistent with unix, recursive creation of directories is not supported.*
- *newDirectoryPath can include paths, as long as all sub directories exist.*

**srmRmdir: (applies to *dir* )**

In:	TUserID	userID,
	TSURLInfo	<u>directoryPath,</u>
	boolean	recursive
Out:	TReturnStatus	<u>returnStatus</u>

**notes:**

- *doRecursiveRemove is false by default.*
- *To distinguish from srmRm(), this function is for directories only.*

**srmRm: (applies to files )**

In:	TUserID	userID,
	TSURLInfo[]	<u>arrayOfFilePaths</u>
Out:	TSURLReturnStatus []	<u>returnStatusOfAllFiles</u>

**notes:**

- *To distinguish from srmRmDir(), this function applies to files only.*

**srmLs: (applies to both *dir* and *file*)**

In:	TUserID	userID,
	TSURLInfo[]	<u>path,</u>
	TFileStorageType	fileStorage Type,
	boolean	fullDetailedList,
	boolean	allLevelRecursive,
	int	numOfLevels,
	int	offset,
	int	count

Out: TMetaDataPathDetail[] details,  
TReturnStatus returnStatus

**notes:**

- *fullDetailedList=false by default.*
  - *For directories, only path is required to be returned.*
  - *For files, path and size are required to be returned.*
- *If fullDetailedList=true, the full details are returned.*
  - *For directories, path and userPermission are required to be returned.*
  - *For files, path, size, userPermission, lastModificationTime, typeOfThisFile, and lifetimeLeft are required to be returned, similar to unix command ls -l.*
- *If allLevelRecursive=true then file lists of all level below current will be provided as well.*
- *If allLevelRecursive is "true" it dominates, i.e. ignore numOfLevels. If allLevelRecursive is "false" or missing, then do numOfLevels. If numOfLevels is "0" (zero) or missing, assume a single level. If both allLevelRecursive and numOfLevels are missing, assume a single level.*
- *When listing for a particular type specified by "fileStorageType", only the files with that type will be in the output.*
- *Empty directories will be returned.*
- *We recommend width first in the listing.*
- *We recommend that list of directories come before list of files in the return array (details).*

**srmMv: (applies to both *dir* and *file*)**

In: TUserID userID,  
TSURLInfo fromPath,  
TSURLInfo toPath

Out: TReturnStatus returnStatus

**notes:**

- *Authorization checks need to be performed on both fromPath and toPath.*

## Data Transfer Functions:

---

**summary:**

**srmPrepareToGet:**

**srmPrepareToPut:**

**srmCopy:**

**srmReleaseFiles: (dir is ok. Will release recursively for dirs)**

**srmRemoveFiles:**

**srmPutDone:**

**srmAbortRequest:**

**srmAbortFiles:**

**srmSuspendRequest:**

**srmResumeRequest:**

**srmStatusOfGetRequest:**

**srmStatusOfPutRequest:**

**srmStatusOfCopyRequest:**

**srmGetRequestSummary:**

**srmExtendFileLifeTime:**

**srmGetRequestID:**

**details:**

**srmPrepareToGet:**

In:	TUserID	<u>userID,</u>
	TGetFileRequest[]	<u>arrayOfFileRequest,</u>
	string[]	<u>arrayOfTransferProtocols,</u>
	string	<u>userRequestDescription,</u>
	TStorageSystemInfo	<u>storageSystemInfo,</u>
	TLifeTimeInSeconds	<u>TotalRetryTime</u>

Out:	TRequestToken	<u>requestToken,</u>
	TGetRequestFileStatus[]	<u>arrayOfFileStatus</u>

**notes:**

- *The userRequestDescription is a user designated name for the request. It can be used in the getRequestID method to get back the system assigned request ID.*
- *Only pull mode is supported.*
- *SRM assigns the requestToken at this time.*
- *Normally this call will be followed by srmRelease().*
- *“retryTime” means: if all the file transfer for this request are complete, then try previously failed transfers for a total time period of “retryTime”.*
- *In case that the retries fail, the return should include an explanation of why the retries failed.*
- *This call is an asynchronous (non-blocking) call. To get subsequent status and results, separate calls should be made.*
- *When the file is ready for the user, the file is implicitly pinned in the cache and lifetime will be enforced.*
- *The invocation of srmReleaseFile() is expected for finished files later on.*

**srmPrepareToPut:**

In:	TUserID	<u>userID</u> ,
	TPutFileRequest[]	<u>arrayOfFileRequest</u> ,
	string[]	<u>arrayOfTransferProtocols</u> ,
	string	<u>userRequestDescription</u> ,
	TOverwriteMode	<u>overwriteOption</u> ,
	TStorageSystemInfo	<u>storageSystemInfo</u> ,
	TLifeTimeInSeconds	<u>TotalRetryTime</u>
Out:	TRequestToken	<u>requestToken</u> ,
	TPutRequestFileStatus[]	<u>arrayOfFileStatus</u>

**notes:**

- *Only push mode is supported for srmPrepareToPut.*
- *StFN (“toSURLInfo” in the TPutFileRequest) has to be local. If stFN is not specified, SRM will name it automatically and put it in the specified user space. This will be returned as part of the “transfer URL”.*
- *srmPutDone() is expected after each file is “put” into the allocated space.*
- *The lifetime of the file starts as soon as SRM get the srmPutDone(). If srmPutDone() is not provided then the files in that space are subject to removal when the space lifetime expires.*
- *“retryTime” is meaningful here only when the file destination is not a local disk, such as tape or MSS.*
- *In case that the retries fail, the return should include an explanation of why the retries failed.*

**srmCopy:**

In:	TUserID	<u>userID</u> ,
	TCopyFileRequest[]	<u>arrayOfFileRequest</u> ,
	string	<u>userRequestDescription</u> ,
	TOverwriteMode	<u>overwriteOption</u> ,
	Boolean	<u>removeSourceFiles</u> (default = false),
	TStorageSystemInfo	<u>storageSystemInfo</u> ,
	TLifeTimeInSeconds	<u>TotalRetryTime</u>
Out:	TRequestToken	<u>requestToken</u> ,
	TCopyRequestFileStatus[]	<u>arrayOfFileStatus</u>

**notes:**

- *Pull mode: copy from remote location to SRM. (e.g. from remote to MSS.)*
- *Push mode: copy from SRM to remote location.*
- *Always release files from source after copy is done.*
- *When removeSourceFiles=true, then SRM will remove the source files on behalf of the caller after copy is done.*
- *In pull mode, send srmRelease() to remote location when transfer is done.*



- *If in push mode, then after transfer is done, notify the caller. User can then release the file. If user releases a file being copied to another location before it is done, then refuse to release.*
- *Note there is no protocol negotiation with the client for this request.*
- *“retryTime” means: if all the file transfer for this request are complete, then try previously failed transfers for a total time period of “retryTime”.*
- *In case that the retries fail, the return should include an explanation of why the retries failed.*
- *When both fromSURL and toSURL are local, perform local copy*
- *Empty directories are copied as well.*

#### **srmRemoveFiles:**

In:	TRequestToken	requestToken,
	TUserID	userID,
	TSURL[]	<u>siteURLs</u>
Out:	TSURLReturnStatus[]	arrayOfReturnStatus

#### **notes:**

- *If requestToken is not provided, then the SRM will do nothing.*
- *It has the effect of a release before the file is removed.*
- *If file is not in cache, do nothing*

#### **srmReleaseFiles:**

In:	TRequestToken	requestToken,
	TUserID	userID,
	TSURL[]	<u>siteURLs,</u>
	Boolean	keepSpace
Out:	TSURLReturnStatus[]	arrayOfReturnStatus

#### **notes:**

- *If requestToken is not provided, then the SRM will release all the files specified by the siteURLs owned by this user, regardless of the requestToken.*
- *If requestToken is not provided, then userID is needed. It may be inferred or provide in the call.*
- *Releasing files will be followed by compacting space, if doDynamicCompactFromNowOn was set to true in a previous srmCompactSpace call.*

#### **srmPutDone:**

In:	TRequestToken	<u>requestToken,</u>
	TUserID	userID,
	TSURL[]	<u>arrayOfSiteURL</u>

Out: TSURLReturnStatus[]      arrayOfReturnStatus

**notes:**

- *Called by user after srmPut()*

**srmAbortRequest:**

In: TRequestToken      requestToken,  
TUserID      userID

Out: TReturnStatus      returnStatus

**notes:**

- *Abort all files in this request regardless of the state. Expired files are released.*

**srmAbortFiles**

In: TRequestToken      requestToken,  
TSURL[]      arrayOfSiteURLs,  
TUserID      userID

Out: TSURLReturnStatus[]      arrayOfReturnStatus

**notes:**

- *Abort all files in this call regardless of the state*

**srmSuspendRequest:**

In: TRequestToken      requestToken  
TUserID      userID

Out: TReturnStatus      returnStatus

**notes:**

- *Suspend all files in this request until srmResumeRequest is issued*

**srmResumeRequest:**

In: TRequestToken      requestToken,  
TUserID      userID

Out: TReturnStatus      returnStatus

**notes:**

- *Resume suspended files in this request*

**srmStatusOfGetRequest:**

In: TRequestToken      requestToken,  
TUserID      userID  
TSURL[]      arrayOfFromSURLs,

Out: TGetRequestFileStatus[]      arrayOfFileStatus

**notes:**

- *If arrayOfFromSURLs is not provided, returns status for all the file requests in this request.*

**srmStatusOfPutRequest:**

In:      TRequestToken      requestToken,  
         TUserID              userID  
         TSURL[]              arrayOfToSURLs,

Out: TPutRequestFileStatus[]      arrayOfFileStatus

**notes:**

- *If arrayOfFromSURLs is not provided, returns status for all the file requests in this request.*

**srmStatusOfCopyRequest:**

In:      TRequestToken      requestToken,  
         TUserID              userID  
         TSURL[]              arrayOfFromSURLs,  
         TSURL[]              arrayOfToSURLs,

Out: TCopyRequestFileStatus[]      arrayOfFileStatus

**notes:**

- *If arrayOfFromSURLs and/or arrayOfToSURLs are not provided, return status for all the file requests in this request.*

**srmGetRequestSummary:**

In:      TRequestToken[]      arrayOfRequestToken,  
         TUserID              userID

Out: TRequestSummary[]      arrayOfRequestSummary  
     TReturnStatus              returnStatus

**srmExtendFileLifeTime:**

In:      TRequestToken      requestToken,  
         TSURL              siteURL,  
         TUserID              userID,  
         TLifeTimeInSeconds      newLifeTime

Out: TReturnStatus              returnStatus,  
     TLifeTimeInSeconds          newTimeExtended

**notes:**

- *newLifeTime is relative to the calling time. Lifetime will be set from the calling time for the specified period.*

- *The number of lifetime extensions maybe limited by SRM according to its policies.*
- *IsExtended = false if SRM refuse to do it. (set newTimeExtended = 0 in this case.)*
- *If original lifetime is longer than the requested one, then the requested one will be assigned.*
- *If newLifeTime is not specified, the SRM can use its default to assign the newLifeTime.*

#### **srnGetRequestID:**

In:	string	<u>userRequestDescription</u> ,
	TUserID	userID
Out:	TRequestTokenReturn[]	arrayOfRequestToken
	TReturnStatus	<u>returnStatus</u>

#### **notes:**

- *If userRequestDescription is null, returns all requests this user has.*
- *If the user assigned the same name to multiple requests, he may get back multiple request IDs each with the time the request was made.*

<b><i>StatusCode specification:</i></b>
---

#### **Note:**

- Status codes represent errors, warnings and status.

<b>Status code</b>	<b>Explanation</b>
--------------------	--------------------

#### **SRM\_SUCCESS:**

- SRM request was successful

#### **Errors:**

#### **SRM\_FAILURE :**

- Requested operation failed for unspecified reason, and additional info is in the explanation string.

#### **SRM\_AUTHENTICATION\_FAILURE:**

- Requester has an invalid authentication information.

#### **SRM\_UNAUTHORIZED\_ACCESS:**

- Requester has no permissions for the operation (although the user could have a valid authentication information).

#### **SRM\_INVALID\_REQUEST:**

- The request is invalid, and additional information may be provided in the explanation string. For example,

- The request token is invalid
- The requested life time of a file is longer than the lifetime of the space.

SRM\_INVALID\_PATH:

- The requested file/directory path is invalid.

SRM\_FILE\_LIFETIME\_EXPIRED:

- The life time on the pinned file has expired

SRM\_SPACE\_LIFETIME\_EXPIRED:

- The life time on the reserved space has expired

SRM\_EXCEED\_ALLOCATION:

- Requester exceeded allocation (number of requests, files or spaces), and the request cannot be placed.

SRM\_NO\_USER\_SPACE:

- The requester does not have enough space to put the file into that space.

SRM\_NO\_FREE\_SPACE:

- SRM has not more space.

SRM\_DUPLICATION\_ERROR :

- Requester tried to create a new file or directory that already exists.

SRM\_NON\_EMPTY\_DIRECTORY:

- Requester tried to remove a non-empty directory without the recursive option set.

SRM\_TOO\_MANY\_RESULTS:

- The request produced too many results; for example, as a result of srmls. The term "too many" is determined by each SRM , and the detailed information, such as the supported max number of results can be returned in the explanation string.

SRM\_INTERNAL\_ERROR:

- SRM has an internal error temporarily. Client may try again.

SRM\_FATAL\_INTERNAL\_ERROR:

- SRM has a severe internal error that cannot be recovered.

SRM\_NOT\_SUPPORTED:

- SRM implementation does not support this functionality that client requested.

### **Status:**

SRM\_REQUEST\_QUEUED

SRM\_REQUEST\_INPROGRESS

SRM\_REQUEST\_SUSPENDEND

SRM \_ABORTED

SRM \_RELEASED

SRM\_FILE\_PINNED

- The requested file is pinned

SRM\_FILE\_IN\_CACHE

- The file is in cache, but not pinned

SRM\_SPACE\_AVAILABLE

- The requested space is reserved and ready to be used

SRM\_LOWER\_SPACE\_GRANTED

- The requested space is not ready, but lower sized space is granted.

SRM\_DONE

SRM\_CUSTOM\_STATUS:

- SRM has a site specific status information. The details are described in the explanation string.

## Appendix:

### SRM WSDL discovery method

May 1, 2003

A) SURL format:

**srm://**host[:port]/[soap\_end\_point\_path?**SFN=**]site\_file\_name

where [...] means optional, and letters in bold are fixed.

We note if the SURL contains the soap\_end\_point\_path, then it is not possible to change the soap endpoint without changing all the previously published SURLs.

Example SURLs:

Without soap\_end\_point\_path:

srm://dm.lbl.gov:4001/ABC/file\_x

with soap\_end\_point\_path:

srm://dm.lbl.gov:4001/srm\_servlet?SFN=ABC/file\_x

B) Given that soap-end-point-path clause is provided, then the soap endpoint is:

https://host[:port]/soap\_end\_point\_path

C) If port is missing, the default port assumed is 8443, which is the port for https with GSI.

The discussion below assumes no endpoint in the SURL, and shows how the soap endpoints and wsdl can be found given an SURL

Issues:

1. We wish to have a way of finding the SRM WSDL for multiple versions from the SURL.
2. We wish to support clients that know what SRM version they want to use. For example, a client that uses version 1.1, should be able to get the WSDL and/or the SOAP endpoint for it directly.
3. We wish to have a default where an SRM version number is not mentioned. The version returned in this case is whatever the SRM currently supports, or if multiple versions are supported, the SRM chooses what to return.
4. We wish to allow a file accessed by a previous SRM version to be accessed by a future SRM version without having to change the SURL. Furthermore, if the file can be accessed by either version simultaneously (that depend on the SRM implementation) that should be possible too.
5. We wish to have a way for a client to find out which version the SRM supports and the endpoint without having to read the WSDL. This is necessary in a changing world, where new version can be introduced.
6. We wish to have a client that can use multiple SRM versions to find out which SRM version is supported by the SRM. This is probably the most realistic scenario, since we cannot expect all SRMs to support the same version at any one time.
7. We wish to have a client find out which SRM versions are supported for accessing a particular file, in case that files can be accessed by multiple SRM versions simultaneously. This is related to point 3 above.

This is a long wish list, but the proposed solution is simple. We assume that the WSDL will contain the version number. First, we propose that every SRM WSDL starts with: SRM version number--> (e.g. <!--SRM version 2.1.3-->)

Now, the solution is as follows:

Give an SURL: srm://host[:port]/path/file (e.g. srm://dm.lbl.gov:4001/ABC/file\_x)

The following can be derived:

Case 1)

For clients that know what SRM versions they want to use:  
<https://host:port/srm/srm.version.wsdl>

`https://host:port/srm/srm.version.endpoint`

For example, given the SURL above, and the client uses version 1.1, you derive:

`https://dm.lbl.gov:4001/srm/srm.1.1.wsdl`

`https://dm.lbl.gov:4001/srm/srm.1.1.endpoint`

Note: the endpoint returned can be any URI, e.g.:

`https://gizmo.lbl.gov:10001/srm/v1.0`

or: `https://dm.lbl.gov:12345/servlet/srm.1.1)`

#### Case 2)

For clients that don't know the version, and want to use the default:

`https://host:port/srm/srm.wsdl`

`https://host:port/srm/srm.endpoint`

For the example above:

`https://dm.lbl.gov:4001/srm/srm.wsdl`

`https://dm.lbl.gov:4001/srm/srm.endpoint`

#### Case 3)

For clients that want to find out the SRM version and endpoint without getting the entire WSDL:

`https://host:port/srm/srm.info`

The srm.info file will contain:

`<!--SRM version number-- --srmEndpoint-->`

For example:

`<!--SRM version 2.1.3-- -- https://gizmo.lbl.gov:10001/srm-->`

#### Case 4)

For servers that support multiple srm version accessing the SAME file:

The same format as above repeating for each srm version.

For example:

`<!--SRM version 1.1-- -- https://sdm.lbl.gov:5005/srm-->`

`<!--SRM version 2.1.3-- -- https://gizmo.lbl.gov:10001/srm-->`

To summarize, the following is what should be supported for WSDL and endpoint discovery:

Given an SURL:

**`srm://host[:port]/site_file_name`**

The following can be derived:

a) `https://host[:port]/srm/srm[.version].wsdl`

b) `https://host[:port]/srm/srm[.version].endpoint`



c) [https://host\[:port\]/srm/srm.info](https://host[:port]/srm/srm.info)

Where the content have the format repeated as many time as there are supported versions:

<!--SRM version number-- --srmEndpoint-->

-----